(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 964 370 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
15.12.1999 Bulletin 1999/50

(51) Int. Cl.^{6}: G07F 7/10, G06K 19/073

(21) Application number: 98110359.1

(22) Date of filing: 05.06.1998

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE
Designated Extension States:
AL LT LV MK RO SI

(71) Applicant:
INTERNATIONAL BUSINESS MACHINES CORPORATION
Armonk, NY 10504 (US)

(72) Inventors:
- Baentsch, Michael
8135 Langnau (CH)
- Höring, Frank
8006 Zurich (CH)
- Buhler, Peter
8800 Thalwil (CH)
- Oestreicher, Marcus
8032 Zurich (CH)
- Eirich, Thomas
8804 Au (CH)

(74) Representative:
Klett, Peter Michael
International Business Machines Corporation,
Säumerstrasse 4
8803 Rüschlikon (CH)

(54) Method and device for loading instruction codes to a memory and linking said instruction codes

(57) A method for loading instruction codes to a first memory and linking said instruction codes is proposed, whereby at least one instruction code has as parameter an address which during a loading step is not determined. This address-parametered instruction code has assigned thereto an address place. A relocation information is loaded which during a linking step effects that the address becomes determined using a starting address and a relative address offset. The then determined address is put at the address place. During the loading step, directly after loading each address-parametered instruction code with its address place, the relocation information is loaded and the address is determined in the linking step.

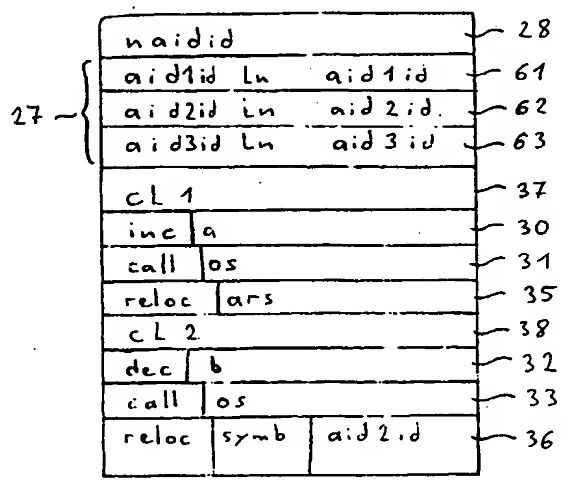


Fig. 2

Description

[0001] The invention relates to a method and a device for loading instruction codes to a memory and linking said instruction codes. More particularly the invention relates to a combined loading and linking method in a resource-constrained environment such as a smartcard, particularly a smartcard offering a Java environment, such as a Javacard.

TECHNICAL FIELD AND BACKGROUND OF THE INVENTION

[0002] When loading and linking machine-dependent, size-efficient code for execution in any resource-constrained code execution environment, in all settings where only parts of the overall executable object code are present during compilation, it is necessary to perform a final step of relocation or linking in which as yet unresolved references to external symbols, e.g., functions, variables, or classes, are replaced by actual addresses valid only in the particular execution context. The linkage information is kept separate from the code in the systems where this approach to code development and installation is commonly used, e.g., personal computers, workstations, or cross-development environments for embedded systems.
[0003] In environments, where the resources computing time, communication bandwidth, and transient memory (RAM) are scarce, and where in addition, writing to persistent memory is much more expensive than writing to temporary memory, and where finally no assumptions about the integrity of the communications infrastructure can be made, new problems appear. In particular, time-efficient ways are to be found to load the code and linkage information into the runtime environment in a secure manner ensuring confidentiality and integrity of the data loaded.
[0004] The initial setting where these assumptions hold true, are smartcards that are to be updated after they have been issued to the customer. In particular, multifunction cards or JavaCards need an efficient resolution of this issue. Therefore, the term 'JavaCard' is used in the following sections inclusively, but not exclusively to denote environments of the nature out-lined above.
[0005] The first problem is the overall time required to load code and linkage information into the smartcard, perform cryptographic decryption and integrity checks on the smartcard over the loaded data, and finally relocate the newly loaded code to prepare it for execution. The second major issue is the amount of temporary data required to perform the above operations.
[0006] In conventional systems, the relocation information comprises a fix-up information and an address info. The fix-up information points to the address place after an instruction code which has an address as parameter which during linking is to be determined. The address information contains an offset and in the case where the relocation type indicates that the address lies in a package different from the package where the address-parametered instruction code lies, a package id, also called "AID", which serves as denominator for starting address, hence designated a package whose starting address is to be used. In the case when the package wherein the address is located, is the same package as where the address-parametered instruction code lies, the starting address is already known as the starting address of this package. The address is then determined as the starting address plus the offset.
[0007] Taking the peculiarities and limitations of smart cards into account, the following basic steps are thus performed to load new executable code into a JavaCard.

- Receive the code into a RAM and transfer it to a persistent memory, e.g. an EEPROM..
- Receive the linkage information (fix-up information and relocation tables) into the RAM and transfer it to persistent memory.
- Optionally, perform decryption and cryptographic integrity checks, commonly known as MACing, of code and linkage information.
- Evaluate the linkage information and for each relocation entry determine the fix-up address and according to the relocation type, compute the actual reference address and write this reference into the appropriate fix-up address in the persistent memory.
- Remove the linkage information from the persistent memory to make room for further code to be loaded in the future.

[0008] This approach however, has the drawback that by performing the cryptographic operations after all data has been loaded, an inefficient number of memory copy operations has to be executed, as those operations can only be run efficiently in RAM.
[0009] Placing linkage information into persistent memory, mainly for reasons of ease of decryption and MACing purposes, a big performance penalty has to be paid, as this data has to be deleted afterwards anyway, which in itself is another expensive write operation on persistent memory.
[0010] By performing relocation after the complete code has been loaded into persistent memory, another performance penalty is incurred, due to that writing single bytes to persistent memory is as expensive in terms of time as writing several bytes, namely a page.

OBJECT AND ADVANTAGES OF THE INVENTION

[0011] It is an object of the invention according to claim 1 and 8, to provide a method and a device for loading instruction codes to a memory and linking said instruction codes, which takes fewer writing cycles to a memory with a relatively high write-access time. This includes the advantage that the result of a completely loaded and linked instruction code sequence is achievable faster and that the lifetime of the device is extended since due to a limited rewritability, the number of write accesses to the same memory is limited also. The effect is even improved by exploiting the effect that a write cycle to an EEPROM for 1 byte takes the same time as does a write cycle for several bytes, i.e. an EEPROM page, e.g. 64 bytes. During the conventional linking procedure, each linking step for each address needs one memory write-access, which with view to the above fact is a waste of access time. The invention exploits to a much higher degree the writing capability of each write-cycle. To this adds that an eventual decryption, using a streaming cipher and/or integrity-check can be performed also without prior writing to the EEPROM.

[0012] The above mentioned objects are met by a method according to claim 1 and a device according to claim 10.

[0013] Furthermore, the amount of data needed for achieving the linking is reduced in size. The amount of data to be sent to the execution environment is smaller as compared to a format where code and relocation information is not interleaved, but strictly separated. In a setting, where first the code and afterwards the relocation information is sent, it is necessary to add to each relocation entry the information for which code address the respective relocation information is valid. This so-called fix-up information is completely made obsolete by the method presented in this disclosure. Net result is a significant reduction in the amount of data sent to the execution environment, in turn leading to a reduction of the overall time necessary to execute the upload process, significantly so, if the communications speed to the execution environment is low.

[0014] Distinction information enabling a distinction between the instruction codes and the relocation information facilitates access to the relocation information. Using code length information which is loaded before each set of code conatining the address-parametered instruction code is advantageous since an immediate information is accessible for recognizing the relocation information. Furthermore the code length information can then be erased, respectively overwritten during the linking process which saves memory space.

[0015] Locating the relative address offset at the address place during the loading step saves memory space because the address place is anyway loaded as a reserved space and is only filled up with insignificant content such as zero values which take the same space as does the address place filled with the offset value.

[0016] It proves of advantage when, in the case when the address to be determined is located in a package which is different from the package the address-parametered instruction code is located in, the starting address is derived form a directly addressable starting address list, because then the usually relatively long AIDs are only once stored and reduced to less space-wasting addresses.

[0017] Memory space is saved when the relocation information and eventual distinction information after the linking step using said relocation information is overwritten by shifting the subsequent instruction codes.

[0018] The first memory where the instruction codes are loaded to and where the linking is done, preferrably has a short write-access-time, such as a RAM which provides for a fast loading and linking procedure. When afterwards the instruction codes are written to a second memory with a longer write-access-time, such as an EEPROM, the usually bigger space of the EEPROM is used to store the whole linked code and the increased write-access time is then no disadvantage since the linking and eventual decryption and integrity-verification already have been performed.

SUMMARY OF THE INVENTION

[0019] A device and method for loading code via a simple protocol into a resource-constrained environment, verify its integrity using cryptographic methods, and relocate the code to transform it into a form actually fit for immediate execution in the above environment, is proposed. The proposed method minimizes the amount of both decryption- and memory-write operations required to ensure a safe transfer and installation of code.

[0020] The main idea consists of interspersing relocation information directly into the code itself and to not cleanly separate these two components. In addition, the presented load format can also efficiently be secured by cryptographic encryption and integrity protection means which can still be checked in extremely resource-constrained execution environments.

[0021] Using a cipher which can be streamed, i.e., which during decryption and MACing only relies on a few bytes of information gathered from processing previous encrypted data and never relies on information only present further forward in the encrypted data stream, enables piecewise decryption which can then be combined with integrity check and linking.

[0022] The invention uses the principle to immediately act upon the linkage information as long as it still resides in RAM and not transfer it to persistent memory until the code is linked. This becomes possible even in a secure manner, once a streaming cipher as suggested above is used.

[0023] Interweaving code and linkage information in a manner that the relocation can immediately be performed in RAM, effects that only fully relocated code segments need be written into persistent memory, i,e, EEPROM and that subsequent writes of single bytes at the fix-up addresses are completely removed.

[0024] The proposed solution is hence interweaving code and linkage information for streaming.

[0025] Memory is divided into immutable memory (e.g., ROM), persistent memory (e.g., EEPROM), and transient memory (e.g., RAM). Only the latter two can be written from a program with access to the memory. Wrting to persistent memory is much more expensive than changing data in transient memory. Writing several bytes, such as pages, in persistent memory is as expensive as writing single bytes. Cryptographic operations require transient memory to run. Transient memory is a scarce resource and heavily contested for by all application components executed on the resource-constrained system.

DESCRIPTION OF THE DRAWINGS

[0026] Examples of the invention are depicted in the drawings and described in detail below by way of example. It is shown in

Fig. 1 an example of an instruction code sequence in an EEPROM according to the state of the art,

Fig. 2 a loading sequence of an instruction sequence interleaved with linking information,

Fig. 3 an arrangement with a virtual machine, a RAM and an EEPROM

[0027] All the figures are for sake of clarity not shown in real dimensions, nor are the relations between the dimensions shown in a realistic scale.

DETAILED DESCRIPTION OF THE INVENTION

[0028] In the following, the various exemplary embodiments of the invention are described.

[0029] In figure 1 an EEPROM 50 is depicted in which a first package P1 between addresses "50" and "90" is stored. The package has an AID "xtra" which is stored at an address "60". The first package P1 is already completely relocated.

[0030] A second package P2 between addresses 100 and 800 comprises a first instruction code 31 at an address 500 which instruction code 31 is a call code followed by a parameter which during loading is zero. The second package P2 comprises a second instruction code 33 at an address 600 which instruction code 33 is a call code followed by a parameter which during loading is also zero.

[0031] The EEPROM 50 further is loaded with a relocation table 47 which comprises relocation information in form of two relocation entries 42, 44. The first relocation entry 42 comprises a first fix-up information which points to the address place 501 where the address as parameter for the first instruction code 31 is to be put. It further comprises a relocation type identifier which here is abbreviated with "ars" for "anonymous-relocation-selfdirected". This identifier is followed by a relative address offset which is here 40. The second relocation entry 44 comprises a second fix-up information which points to the address place 601 where the address as parameter for the second instruction code 33 is to be put. It further comprises a relocation type identifier which here is abbreviated with "symb" for symbolic relocation.

[0032] This identifier is followed by a relative address offset which is here 20 and by an application id "xtra".

[0033] A package list 29 comprises a list of all already linked packages, in this case for the first package P1 one entry which tells that the starting address of that package is "50".

[0034] According to the state of the art, on a smartcard, except for the application id table 29, all the above information is loaded into a RAM first and from there to the EEPROM. This is done due to the fact that on a smartcard the RAM is much smaller than the EEPROM. For linking, the whole above infromation is needed which leads to the obligation to have the whole information accessible at once. This can only be guarantueed for the EEPROM.

[0035] In the case, the code is encrypted, a decryption operation follows. For that, the code is piecewise reloaded to the RAM where a decryption is conducted and the decrypted code is reloaded to the EEPROM. In case, an integrity check is to be done, the code is another time piecewise reloaded to the RAM and the integrity check is performed, resulting in an integrity check vector, signalizing if the code has passed the check or not.

[0036] Then follows the linking procedure which again can only be performed in the RAM. Hence, part of the relocation table 47 is read into the RAM and the linking procedure starts.

[0037] The linking procedure goes through the relocation table 47 and starts with the first relocation entry 42. The entry is processed in that due to the fact that the relocation type is selfdirected, as the starting address the starting address of the second package P2, i.e. the package which is currently loaded, i.e. "100", is taken. To this starting address the offset "40" given in the first relocation entry 42 is added, thereby arriving at a definitive determined address "140". This value is entered as the determined address in the address place 501 of the first instruction code 31.

[0038] Then the next relocation entry is processed in that due to the fact that the relocation type is symbolic, the starting address is derived in that the entries in the package list 29 are checked one by one and each package is addressed for its therein stored AID and in the case of a matching AID the therewith identified package starting address is taken as the starting address for the determination of the address for the second instruction code 33. Since the first package P1 has the AID "xtra", it is recognized as being the right package.

[0039] This operation results here hence in the starting address "50". To this starting address the offset "20" given in the second relocation entry 44 is added, thereby arriving at a definitive determined address "70" in the first package P1. This value is entered as the determined address in the address place of the second instruction code 33, since the address place pointed to by the second relocation entry 44 is 601.

[0040] Therewith the linking procedure is completed and the relocation table 47 and the package list 29 are no longer needed.

[0041] The above procedure has various drawbacks: The big number of writing cycles to and from the EEPROM on one hand is time-consuming because EEPROMs have longer access times than RAMs and on the other hand, the lifetime of the system is reduced because the number of rewriting cycles of EEPROMs is limited.

[0042] In figure 2, the loading sequence of instruction codes and relocation information, also called load file, according to the invention is depicted. First, an application-id id list 27 is transferred which is preceded by an application-id id list length information 28, short naidid. The application-id id list 27 has three entries, a first application-id id aid1id, a second application-id id aid2id and a third application-id id aid3id.

[0043] Then follows a first distinction information 37 in form of a code length information cl1. An incrementing instruction code "inc" with a parameter a follows, before the first address-parametered instruction code 31 "call" with a parameter os. Then follows already the first relocation information 35 which belongs to the first instruction code 31.

[0044] Then follows a second distinction information 38 in form of a code length information cl2. A decrementing instruction code "dec" with a parameter a follows, before the second address-parametered instruction code 33 "call" with a parameter os. Then follows already the second relocation information 36 which belongs to the second instruction code 31.

[0045] Additionally to the load file so-called load-file metainformation such as code length, decryption key, MAC key a.s.o. can be loaded.

[0046] Hence, the relocation information list 47 no longer exists as a separate block but is split up and interwoven with the instruction code sequence. The fact that the relocation information 35, 36 is loaded directly behind its instruction code 31, 33, renders obsolete the address information which in the linking procedure according to the state of the art was needed for pointing to the assigned address place. The assigned address place is here automatically recognized due to the fixed spatial relation between the place of the relocarion information 35, 36 and place of the instruction code 31, 33. Hence, for each relocation step, two bytes are saved which reduces the amount of total bytes to be loaded and transferred between the memories.

[0047] Also, since the address places behind the instruction codes 31, 33 are not filled with zero values in the loading procedure but contain the values of the relative address offsets os, again two bytes per relocation information 35, 36 are saved. The system simply needs to know that the relative address offset os is to be found at the asssigned address place and not behind the relocation information 35, 36. The selfdirected relocation type hence is reduced to a 1 byte relocation instruction 35.

[0048] The distinction information 37, 38 is used to enable a distinction between the instruction codes 30, 31, 32, 33 and the relocation information 35, 36. This is used to thereby detect which part of the code is the relocation information 35, 36and is hence to be treated accordingly.

[0049] The interleaving of the relocation information 35, 36 with the instruction code sequence is now used to relocate the address right after loading. The availability of the relocation information 35, 36 already before the whole loading sequence is loaded, renders possible an immediate relocation.

[0050] Immediate means here exemplarily, that the RAM 51 is either filled or that the end of the code sequence has been reached. Before relocation, an eventual decryption and/or integrity check can be done in the case, the code has been loaded in encrypted form and/or an integrity check is needed or desired. Such decryption can be done with a streamed cipher which makes the piece of code that fills the RAM 51 decryptable without relying on information only available from subsequent code.

[0051] Relocation is started which means that the first instruction code 31 is linked to the determined address and afterwards the respective relocation information 35 is obsolete and can be overwritten. This can be done by shifting the subsequent code upwards by the right number of bytes. Therefor again the code length information can be used which saves space in the EEPROM 50.

[0052] Application-ids as defined by the ISO 7816 standard and commonly used in state of the art implementations of loading & linking new code on a JavaCard, are relatively long, i.e., between 5 and 15 bytes. The application-id id list 27 provides for a mechanism that reduces the required amount of both data storage during link time as well as that of data transmitted during the loading stage.

[0053] In order to facilitate these reductions, as a first stage in the loading process, the number of AIDs against which the following code is to be linked, is transmitted to the arrangement, e.g. a smartcard. Now the smartcard can allocate (transient) data storage for the same number of addresses as AIDs thus announced.

[0054] This data storage is subsequently filled with the start addresses of the different packages identified by said AIDs that are transmitted subsequently to the smartcard. The runtime environment therefor provides a method to look up the start addresses of the packages with the AIDs as received into the smartcard. The therefrom resulting start addresses are then entered into the AID-id table 27. In summary, the AIDs are thus only sent once to the smart-card in some predefined order which then is reflected in the AID-id table 27 thus established. This facilitates the use of only short (1 byte) AID ids, as opposed to the long 5-15 bytes AIDs, in any symbolic link information contained in the subsequently sent code and relocation information.

[0055] Since the encryption, integrity-check and linking can be performed piecewise, i.e. for only a part of the loaded code, one part following the other, all operations can be done right after loading the respective part to the RAM 51 and then loading the decrypted, integrity-verified and linked code to the EEPROM.

[0056] The reduced size of the RAM 51 is hence of lower impact on the loading and linking process.

[0057] Furthermore, a padding process can be introduced, i.e. filling up empty space in the RAM 51 with zero values in order to process only complete information comprising the address-parametered instruction code 31, 33 with the thereto belonging relocation information 35, 36. This is to avoid the necessity to store intermediately instruction codes whose relocation information did not fit into the RAM 51 anymore. If no relocation information is contained in the RAM 51 after one loading step which fills up the RAM 51, no linking is needed and the RAM 51 content can be transferred directly to the EEPROM 50.

[0058] In figure 3 an arrangement is depicted which is a resource-constrained environment such as particularly existing on smartcards. The restriction expresses itself mainly in the size of the RAM 51 and the EEPROM 50, but also in small computing power which is relevant for cryptography, small bus width, which is relevant for intercommunication speed, small clock cycles which leads to small inter-component communication speed and low processing speed, etc.

[0059] A microprocessor µP 1 runs a virtual machine 10 which is connected to a first memory 51, which here is a memory with a short write-access time, here a RAM and a second memory 50 which has a relatively longer write-access time, here an EEPROM.

[0060] The RAM 51 is further connected to a coprocessor cµP 2 and to a protocol-handling unit PHU 15 which handles APDU traffic arriving at and leaving the depicted arrangement which may be integrated on a smartcard.

[0061] Via the PHU 15, code is loaded in to the RAM 51. This code comprises the instruction code sequence and the relocation information 35, 36. In the prior art, the whole code sequence for an applet is loaded via the RAM 51 into the EEPROM 50. Only then, apart from eventual decryption and/or integrity-check, the linking procedure is performed.

[0062] The RAM 51 is smaller than the EEPROM 50 such that the code sequence can only piece-wise be transmitted to the EEPROM 50, each piece fitting into the RAM 51. With the herein described new method, each piece is linked completely before it is loaded into the EEPROM 50. With the overwriting feature, even less space in the EEPROM 50 is used which extends again its lifetime. The code loading between the memories 50, 51 should be done via a transaction-subsystem to ensure integrity of the EEPROM 50 in case of errors during the loading, linking and installation of new code on the smartcard. Decryption and integrity check is preferrably done by the coprocessor cµP 2, while the loading and linking is preferrably done by the microprocessor µP 1.

[0063] Instead of call instruction codes any other address-parametered instruction codes can be used. Other parameters like the numbers of the addresses are for sake of example only and can be varied without leaving the scope of the invention.

Claims

1. Method for loading instruction codes (30, 31, 32, 33) to a first memory (51) and linking said instruction codes (30, 31, 32, 33), whereby at least one instruction code (31, 33) has as parameter an address which during a loading step is not determined and whereby said address-parametered instruction code (31, 33) has assigned thereto an address place and whereby for said address-parametered instruction code (31, 33) a relocation information (42, 44, 35, 36) is loaded which during a linking step effects that said address becomes determined using a starting address (xtra, aid1id, aid2id, aid3id) and a relative address offset (os) and whereby said determined address is put at said address place, characterized in that during said loading step, directly after loading each said address-parametered instruction code (31, 33) with its address place, said relocation information (42, 44, 35, 36) is loaded and said address is determined in said linking step.

2. Method according to claim 1, characterized in that distinction information (37, 38) is loaded which enables a distinction between the instruction codes (30, 31, 32, 33) and the relocation information (42, 44, 35, 36),

3. Method according to claim 2, characterized in that the distinction information (37, 38) comprises code length information (cl1, cl2) which is loaded before each of the address-parametered instruction codes (31, 33).

4. Method according to one of claims 1 to 3, characterized in that during the loading step the relative

address offset (os) is located at the address place.

5. Method according to one of claims 1 to 4, characterized in that in the case when the address to be determined is located in a package (P1) which is different from the package (P2) the address-parametered instruction code (31, 33) is located in, the starting address (xtra, aid1id, aid2id, aid3id) is derived form an directly addressable starting address list (27).

6. Method according to claim 5, characterized in that the starting address list (27) is obtained by loading denominators for the starting addresses (xtra, aid1id, aid2id, aid3id) and by performing by use of a table a subsequent lookup step which results in said starting addresses (xtra, aid1id, aid2id, aid3id) and by substituting said denominators by said starting addresses (xtra, aid1id, aid2id, aid3id).

7. Method according to one of claims 1 to 6, characterized in that the relocation information (35, 36) and eventual distinction information (37, 38), after the linking step using said relocation information (35, 36), is overwritten by shifting the subsequent instruction codes (30, 31, 32, 33).

8. Method according to one of claims 1 to 7, characterized in that the first memory (51) where the instruction codes (30, 31, 32, 33) are loaded to and where the linking is done, has a short write-access-time, such as a RAM, and that afterwards the instruction codes (30, 31, 32, 33) are written to a second memory (50) with a longer write-access-time, such as an EEPROM.

9. Method according to one of claims 1 to 8, characterized in that the instruction codes (30, 31, 32, 33) are loaded in an encrypted form and/or with an integrity check information and that said instruction codes (30, 31, 32, 33) are decrypted and/or checked upon their integrity in said first memory (51) with short write-access-time.

10. Device with a first memory (51) and with loading means for loading to said first memory (51) instruction codes (30, 31, 32, 33) and having linking means for linking said instruction codes (30, 31, 32, 33), whereby at least one instruction code (31, 33) has as parameter an address which during loading is not determined and whereby said address-parametered instruction code (31, 33) has assigned thereto an address place and whereby for said address-parametered instruction code (31, 33) with said loading means a relocation information (42, 44, 35, 36) is loadable which during linking is able to effect that said address becomes determined using a starting address (xtra, aid1id, aid2id, aid3id) and a relative address offset (os) and whereby said determined address is putable at said address place, characterized in that during loading, directly after loading each said address-parametered instruction code (31, 33) with its address place, said relocation information (42, 44, 35, 36) is loadable and said address is determinable during linking.

11. Device according to claim 10, characterized in that the first memory (51) where the instruction codes (30, 31, 32, 33) are loadable to and where the linking is performable, has a short write-access-time, e.g. being a RAM, and that said device further comprises a second memory (50) whereto afterwards the instruction codes (30, 31, 32, 33) are writable, said second memory (50) having a longer write-access-time, e.g. being an EEPROM.

12. Device according to claim 10 or 11, characterized in that it comprises decryption means for decrypting the instruction codes (30, 31, 32, 33) which are loaded in an encrypted form and/or comprises integrity-checking means for checking the integrity of at least said instruction codes (30, 31, 32, 33) which are loaded with integrity check information.

13. Device according to one of claims 10 to 12 and/or for carrying out a method according to one of claims 1 to 9, characterized in that it comprises a smartcard, particularly a Javacard, or an electronic circuitry therefor.

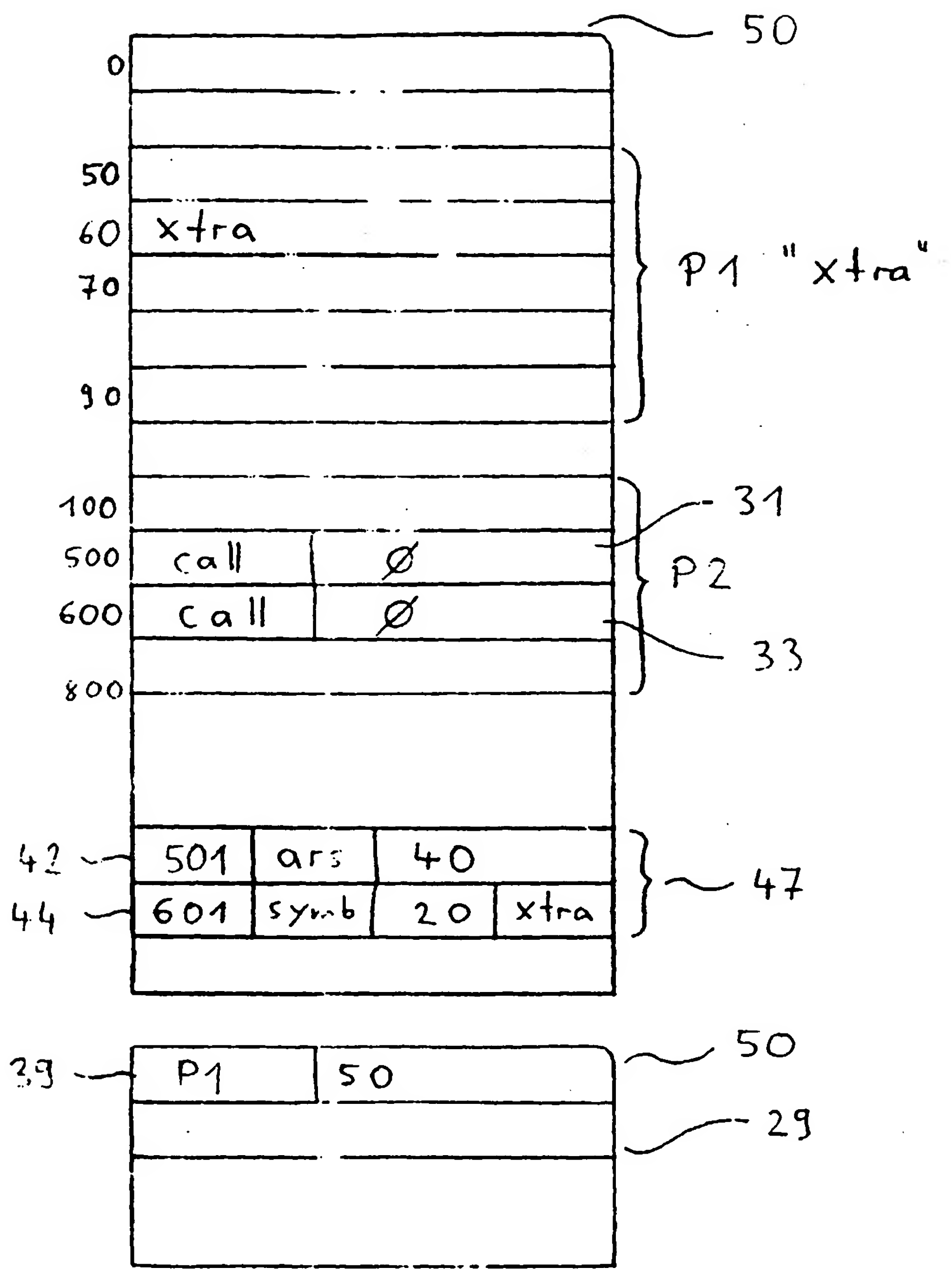
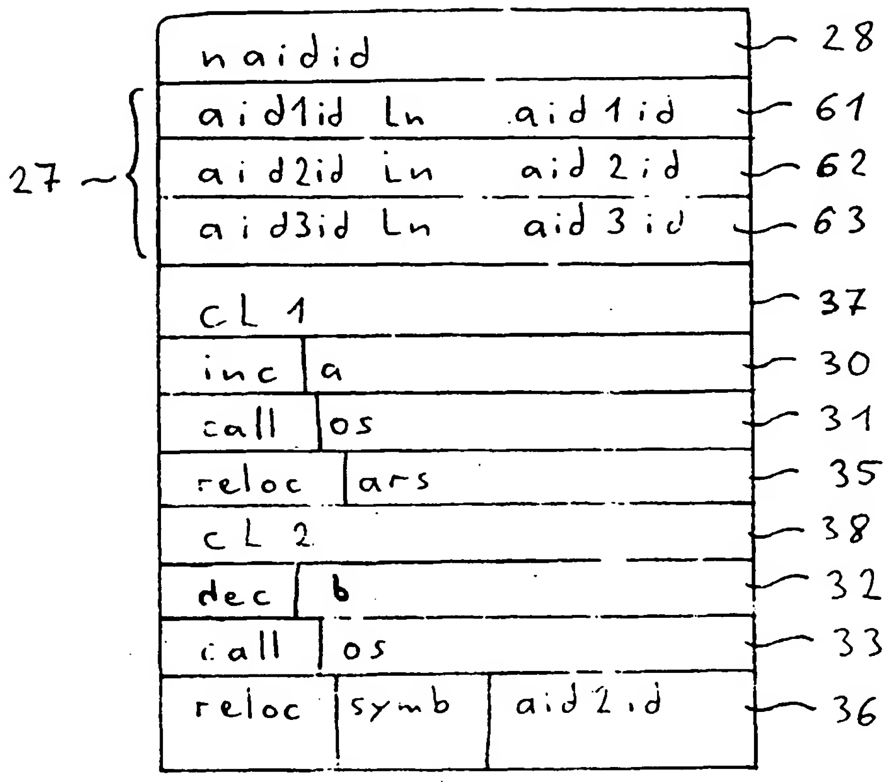
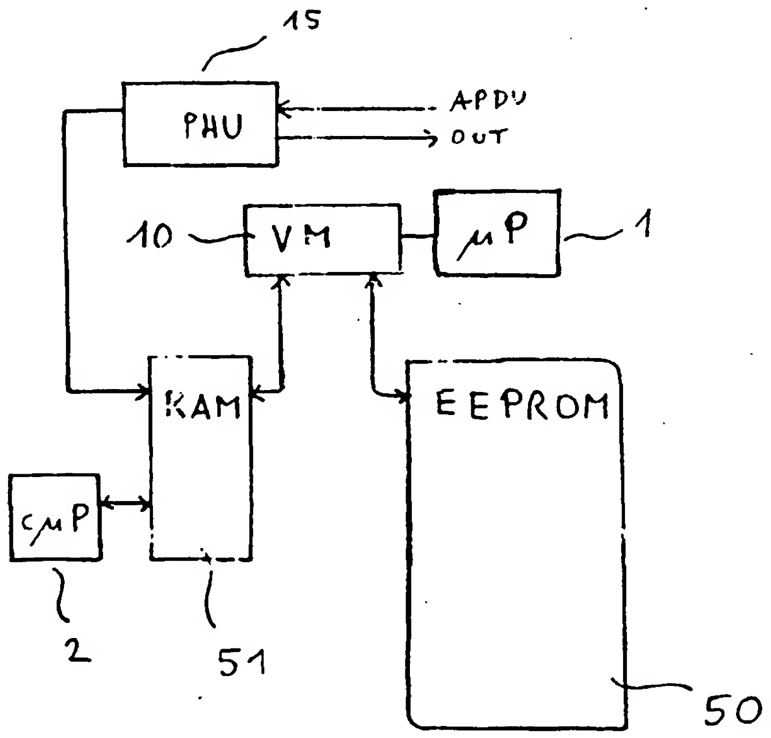


Fig. 1 Prior Art



naidid
aid1id Ln aid1id
aid2id Ln aid2id
aid3id Ln aid3id
cL 1
inc a
call os
reloc ars
cL 2
dec b
call os
reloc symb aid2id
27
28
61
62
63
37
30
31
35
38
32
33
36

Fig. 2



15
PHU
APDU
OUT
10
VM
μP
1
RAM
EEPROM
CμP
2
51
50

Fig. 3

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 98 11 0359

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	EP 0 383 518 A (HITACHI MAXELL) 22 August 1990 * abstract; claims; figures * * column 1, line 53 - column 3, line 52 * ---	1,3,8, 10,11,13	G07F7/10 G06K19/073
A	US 4 829 169 A (H. WATANABE) 9 May 1989 ---		
A	EP 0 563 997 A (TOSHIBA) 6 October 1993 -----		

TECHNICAL FIELDS SEARCHED (Int.Cl.6)
G07F
G06K

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	3 March 1999	David, J

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons
& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 98 11 0359

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report.
The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

03-03-1999

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0383518 A	22-08-1990	JP 2214994 A	27-08-1990
		DE 69013026 D	10-11-1994
		DE 69013026 T	18-05-1995
		US 5038025 A	06-08-1991
US 4829169 A	09-05-1989	JP 62231393 A	09-10-1987
EP 0563997 A	06-10-1993	JP 5282857 A	29-10-1993
		JP 5290567 A	05-11-1993
		JP 5313989 A	26-11-1993
		KR 9701201 B	29-01-1997
		US 5745912 A	28-04-1998

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82